FIG. 1

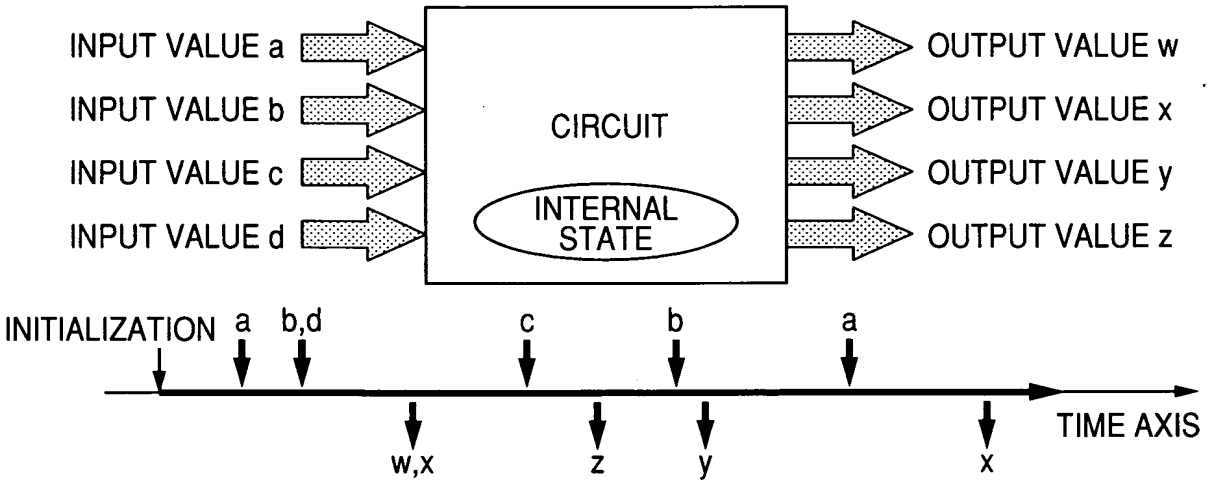

INPUT VALUE a
INPUT VALUE b
INPUT VALUE c
INPUT VALUE d
CIRCUIT
INTERNAL STATE
OUTPUT VALUE w
OUTPUT VALUE x
OUTPUT VALUE y
OUTPUT VALUE z
INITIALIZATION
a
b,d
c
b
a
w,x
z
y
x
TIME AXIS

FIG. 2

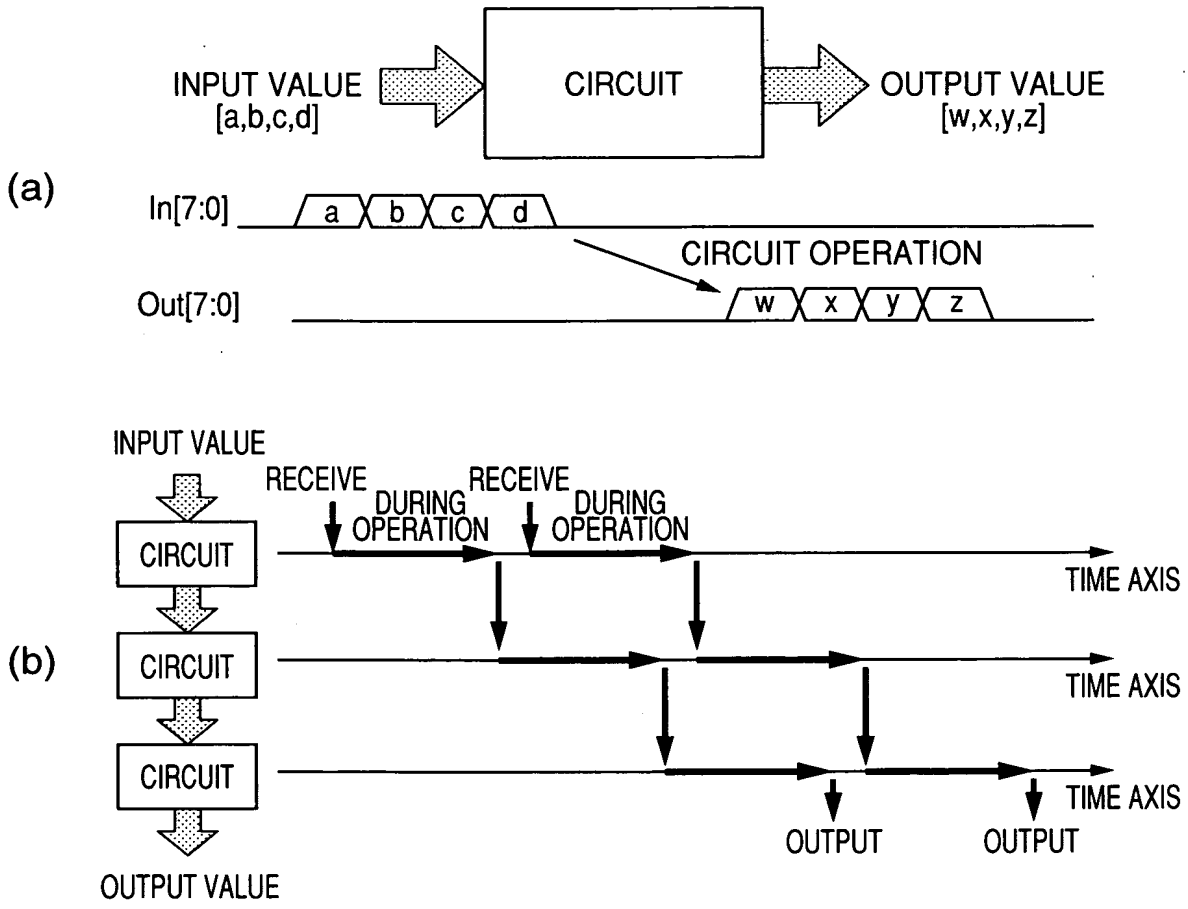

INPUT VALUE
[a,b,c,d]
CIRCUIT
OUTPUT VALUE
[w,x,y,z]
(a)
In[7:0]
a
b
c
d
CIRCUIT OPERATION
Out[7:0]
w
x
y
z
INPUT VALUE
RECEIVE
DURING OPERATION
RECEIVE
DURING OPERATION
CIRCUIT
CIRCUIT
CIRCUIT
(b)
TIME AXIS
TIME AXIS
TIME AXIS
OUTPUT
OUTPUT
OUTPUT VALUE

FIG. 3

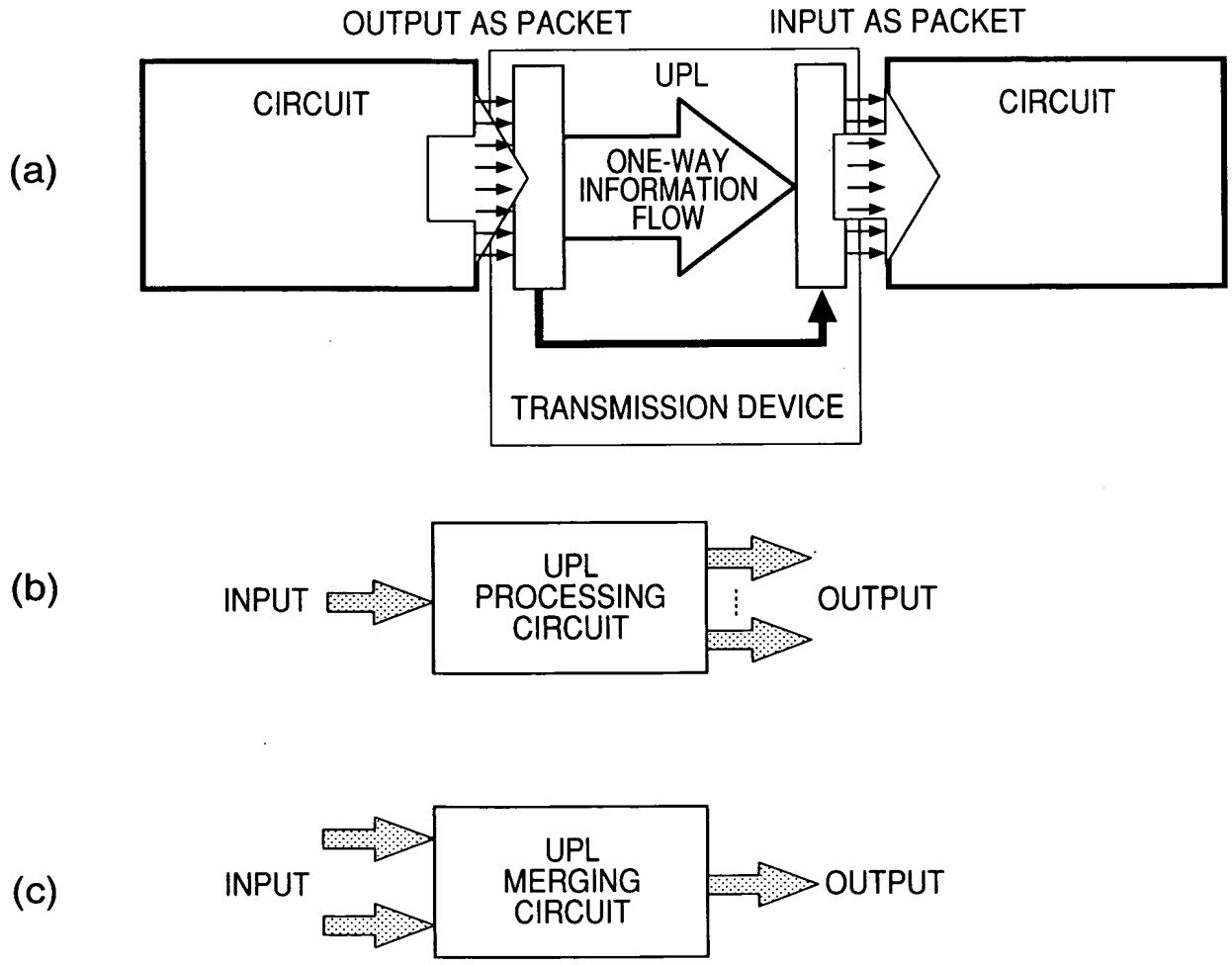

(a)
OUTPUT AS PACKET
INPUT AS PACKET
CIRCUIT
UPL
ONE-WAY INFORMATION FLOW
CIRCUIT
TRANSMISSION DEVICE
(b)
INPUT
UPL PROCESSING CIRCUIT
OUTPUT
(c)
INPUT
UPL MERGING CIRCUIT
OUTPUT

FIG. 4

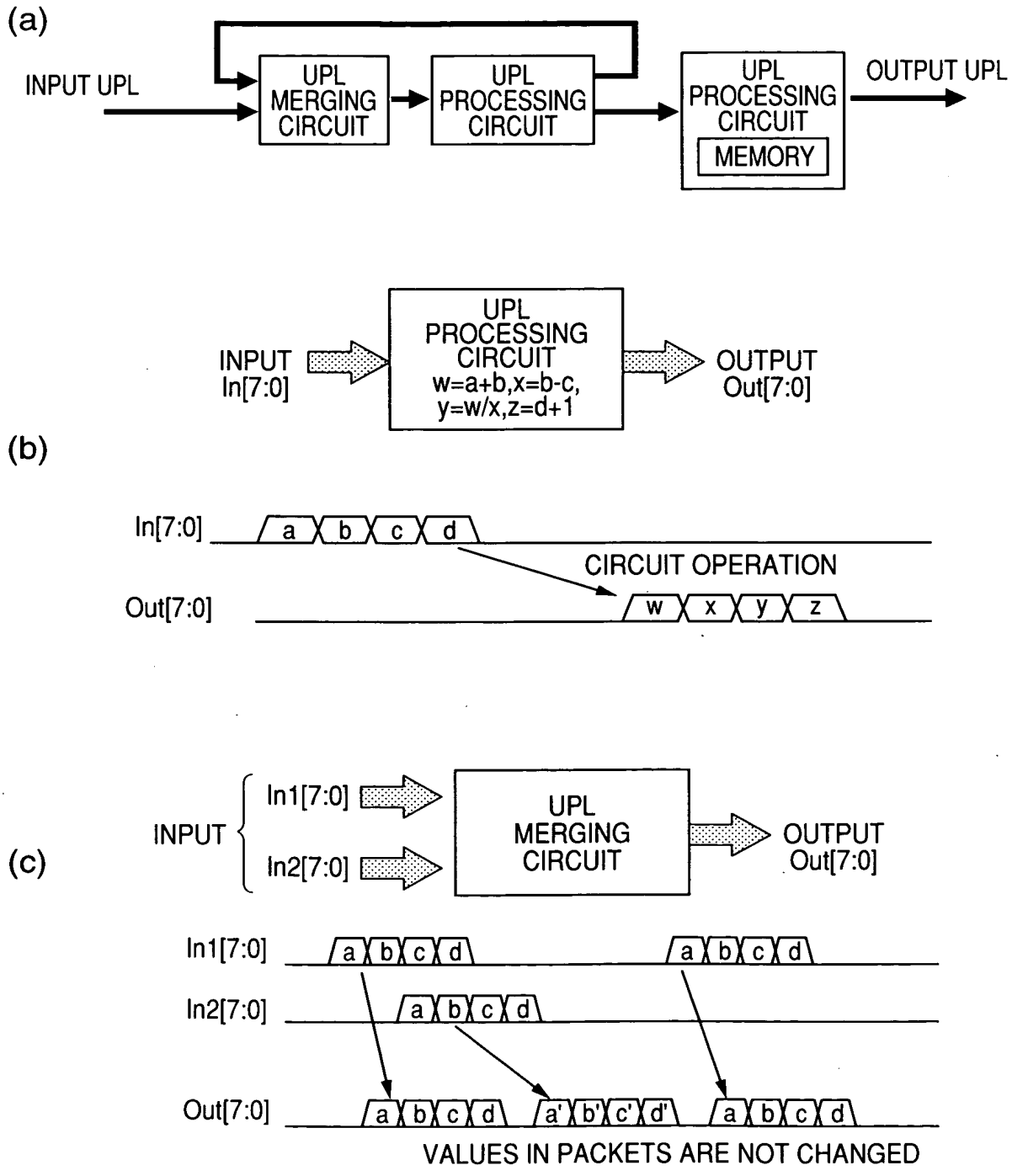

(a)
INPUT UPL
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT
UPL PROCESSING CIRCUIT
MEMORY
OUTPUT UPL
INPUT In[7:0]
UPL PROCESSING CIRCUIT w=a+b,x=b-c, y=w/x,z=d+1
OUTPUT Out[7:0]
(b)
In[7:0]
a b c d
CIRCUIT OPERATION
Out[7:0]
w x y z
(c)
INPUT
In1[7:0]
In2[7:0]
UPL MERGING CIRCUIT
OUTPUT Out[7:0]
In1[7:0]
In2[7:0]
Out[7:0]
a b c d
a' b' c' d'
VALUES IN PACKETS ARE NOT CHANGED

FIG. 5

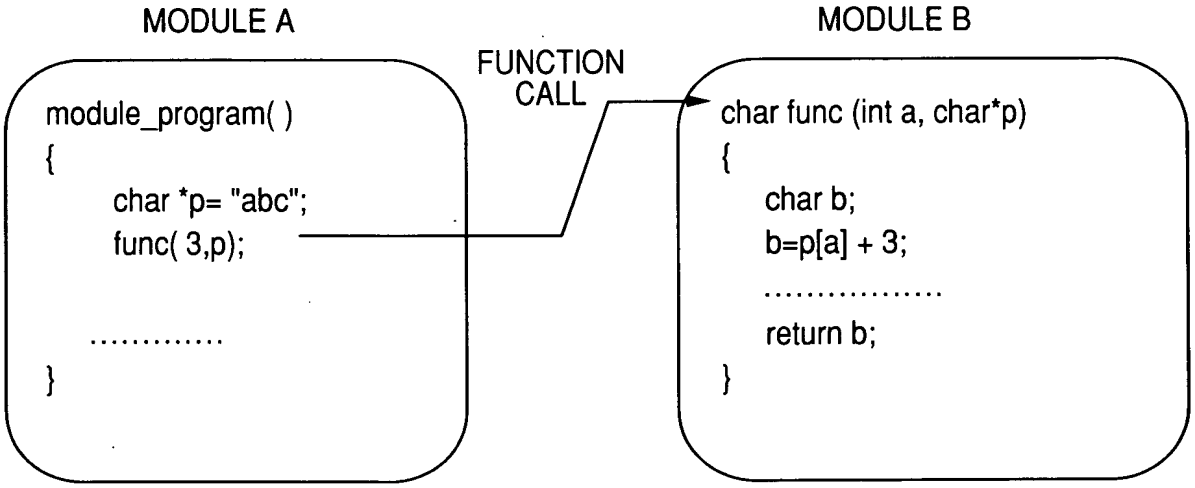

MODULE A
FUNCTION CALL
MODULE B
module_program( )
{
char *p= "abc";
func(3,p);
.....
}
char func (int a, char*p)
{
char b;
b=p[a] + 3;
.....
return b;
}

FIG. 6

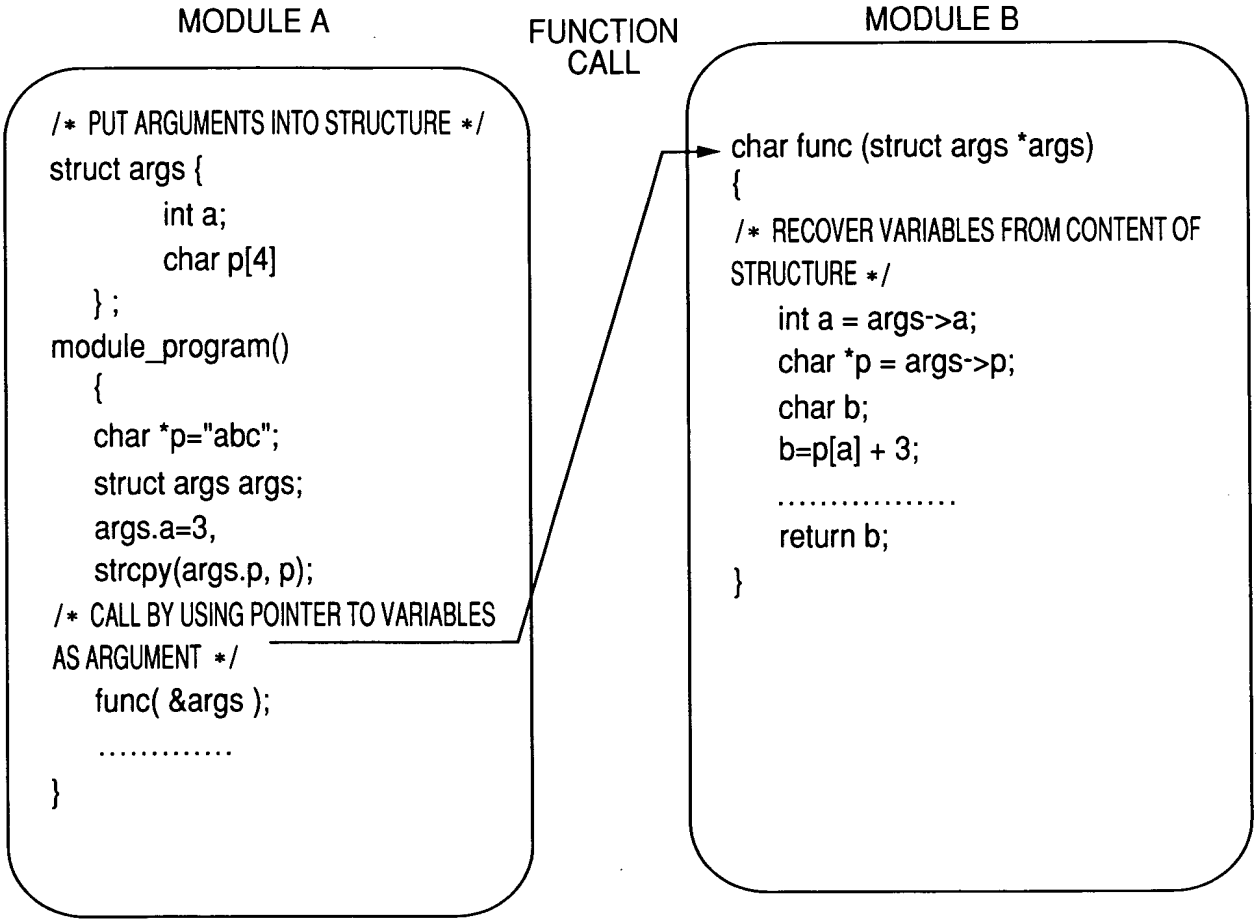

MODULE A
FUNCTION CALL
MODULE B
/* PUT ARGUMENTS INTO STRUCTURE */
struct args {
int a;
char p[4]
} ;
module_program()
{
char *p="abc";
struct args args;
args.a=3,
strcpy(args.p, p);
/* CALL BY USING POINTER TO VARIABLES AS ARGUMENT */
func(&args);
.....
}
char func (struct args *args)
{
/* RECOVER VARIABLES FROM CONTENT OF STRUCTURE */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}

FIG. 7

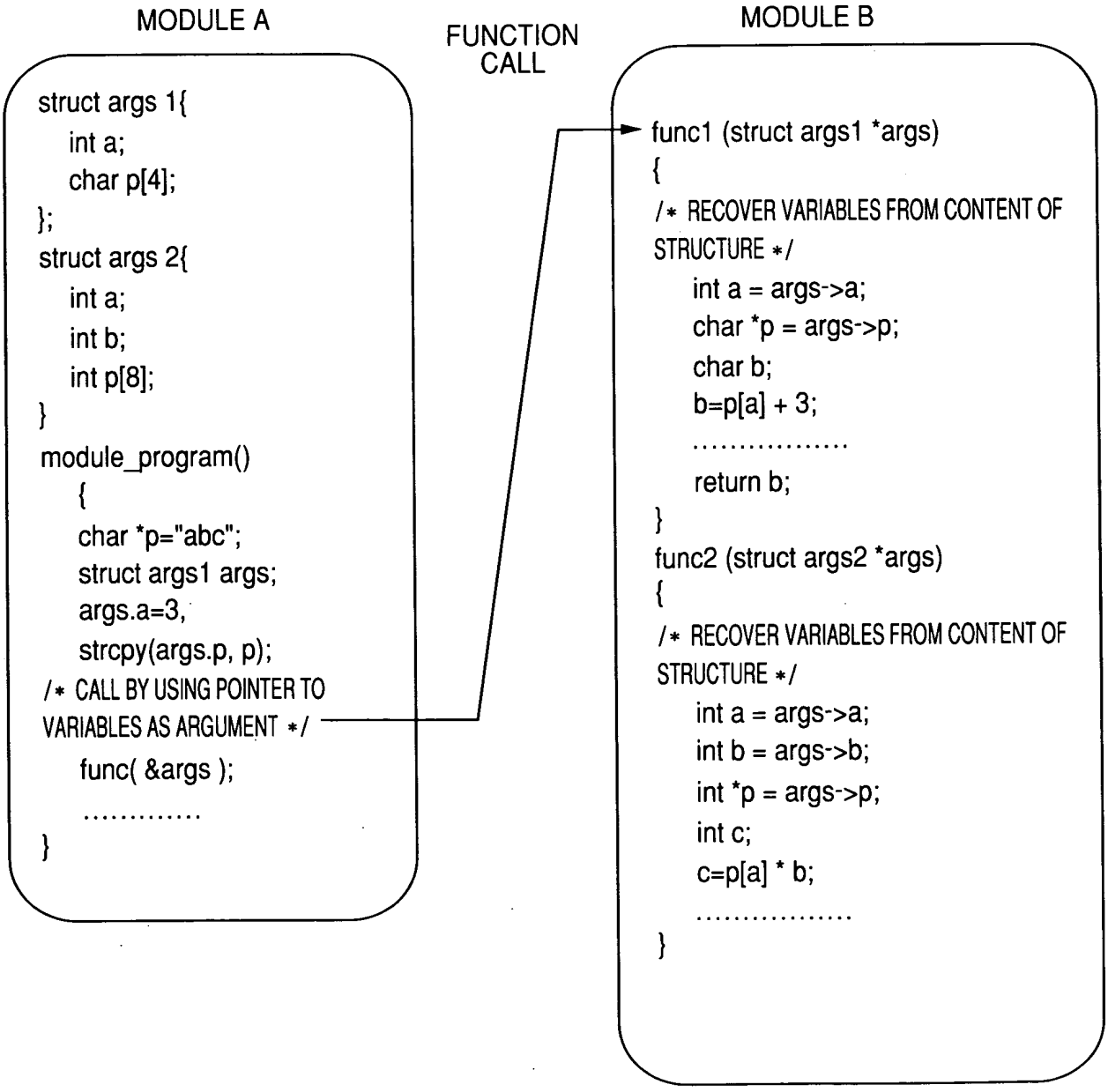

MODULE A
FUNCTION CALL
MODULE B
struct args 1{
int a;
char p[4];
};
struct args 2{
int a;
int b;
int p[8];
}
module_program()
{
char *p="abc";
struct args1 args;
args.a=3,
strcpy(args.p, p);
/* CALL BY USING POINTER TO VARIABLES AS ARGUMENT */
func(&args);
.....
}
func1 (struct args1 *args)
{
/* RECOVER VARIABLES FROM CONTENT OF STRUCTURE */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args2 *args)
{
/* RECOVER VARIABLES FROM CONTENT OF STRUCTURE */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

FIG. 8

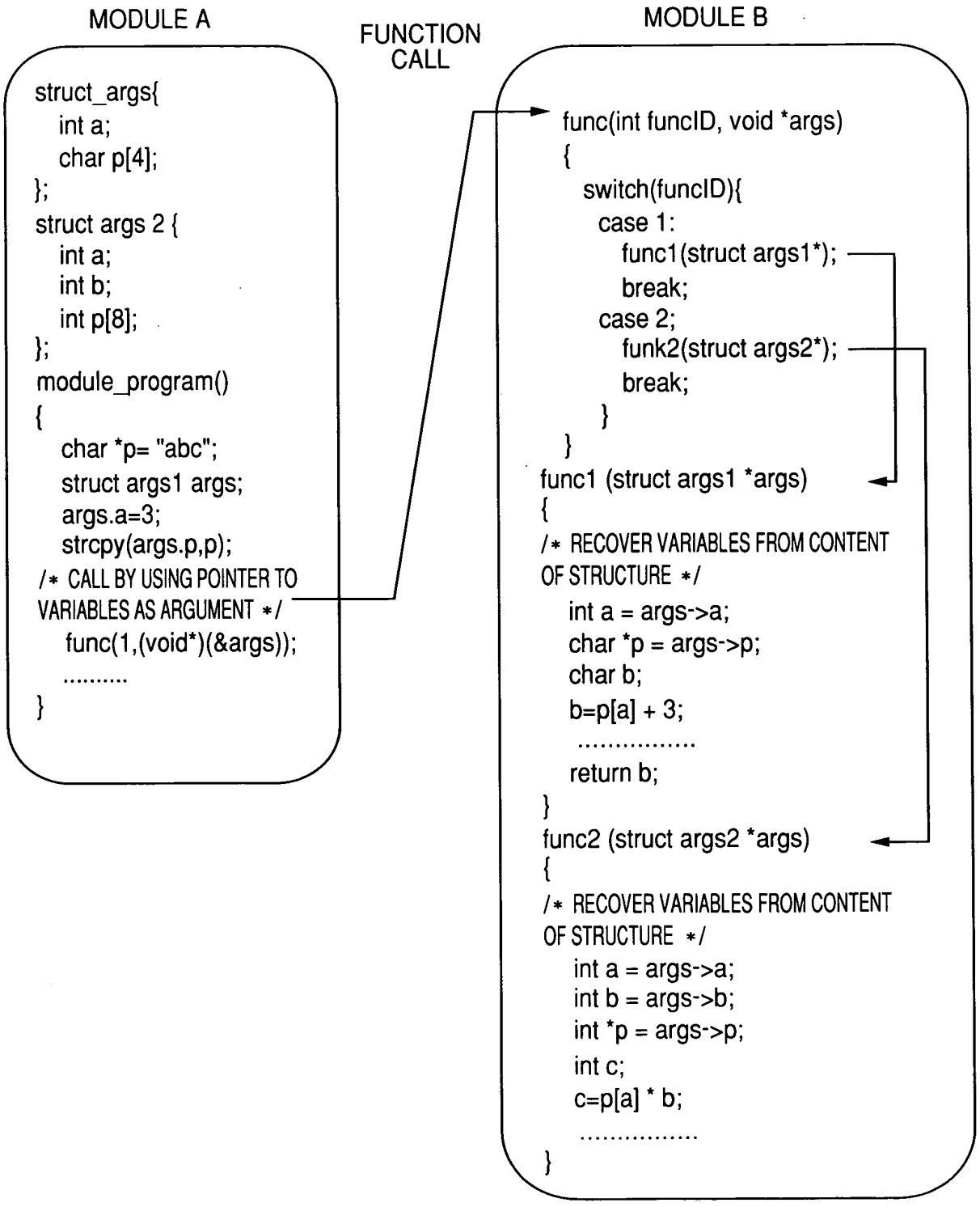

MODULE A
FUNCTION CALL
MODULE B
struct_args{
int a;
char p[4];
};
struct args 2 {
int a;
int b;
int p[8];
};
module_program()
{
char *p= "abc";
struct args1 args;
args.a=3;
strcpy(args.p,p);
/* CALL BY USING POINTER TO VARIABLES AS ARGUMENT */
func(1,(void*)(&args));
.....
}
func(int funcID, void *args)
{
switch(funcID){
case 1:
func1(struct args1*);
break;
case 2;
funk2(struct args2*);
break;
}
}
func1 (struct args1 *args)
{
/* RECOVER VARIABLES FROM CONTENT OF STRUCTURE */
int a = args->a;
char *p = args->p;
char b;
b=p[a] + 3;
.....
return b;
}
func2 (struct args2 *args)
{
/* RECOVER VARIABLES FROM CONTENT OF STRUCTURE */
int a = args->a;
int b = args->b;
int *p = args->p;
int c;
c=p[a] * b;
.....
}

FIG. 9

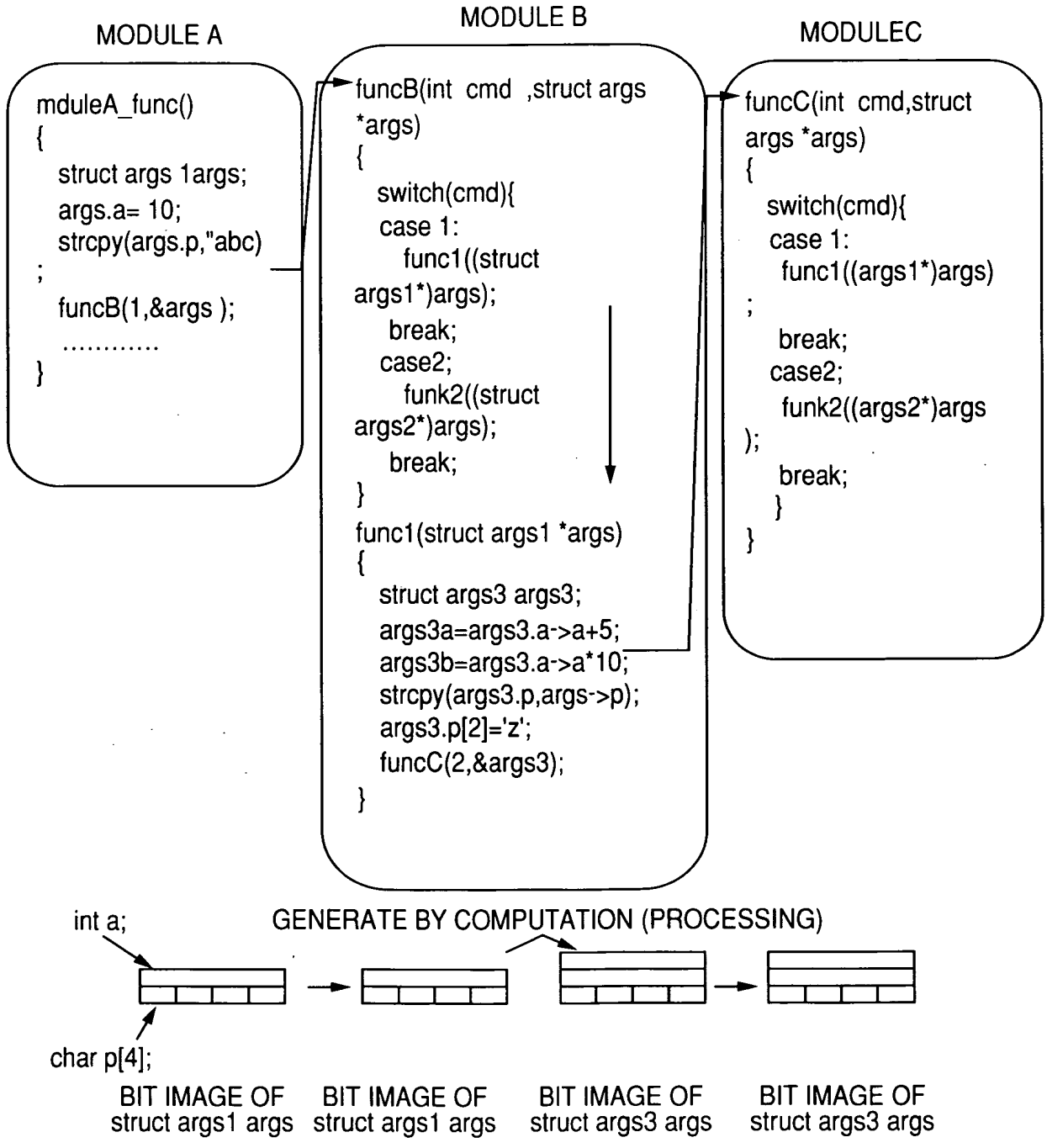

MODULE A
mduleA_func()
{
struct args 1args;
args.a= 10;
strcpy(args.p,"abc)
;
funcB(1,&args);
.....
}
MODULE B
funcB(int cmd ,struct args *args)
{
switch(cmd){
case 1:
func1((struct args1*)args);
break;
case2;
funk2((struct args2*)args);
break;
}
func1(struct args1 *args)
{
struct args3 args3;
args3a=args3.a->a+5;
args3b=args3.a->a*10;
strcpy(args3.p,args->p);
args3.p[2]='z';
funcC(2,&args3);
}
MODULEC
funcC(int cmd,struct args *args)
{
switch(cmd){
case 1:
func1((args1*)args)
;
break;
case2;
funk2((args2*)args
);
break;
}
}
int a;
GENERATE BY COMPUTATION (PROCESSING)
char p[4];
BIT IMAGE OF struct args1 args
BIT IMAGE OF struct args1 args
BIT IMAGE OF struct args3 args
BIT IMAGE OF struct args3 args

FIG. 10

```
func1(struct args1 *args)
{
        struct args3 args3;
        args3.a = args3.a->a+5;
        args3.b = args3.a->a*10;
        strcpy(args3.p,args->p);
        args3.p[2] = 'z';
        funcC( 1,(void*)( &args3) );
}
```

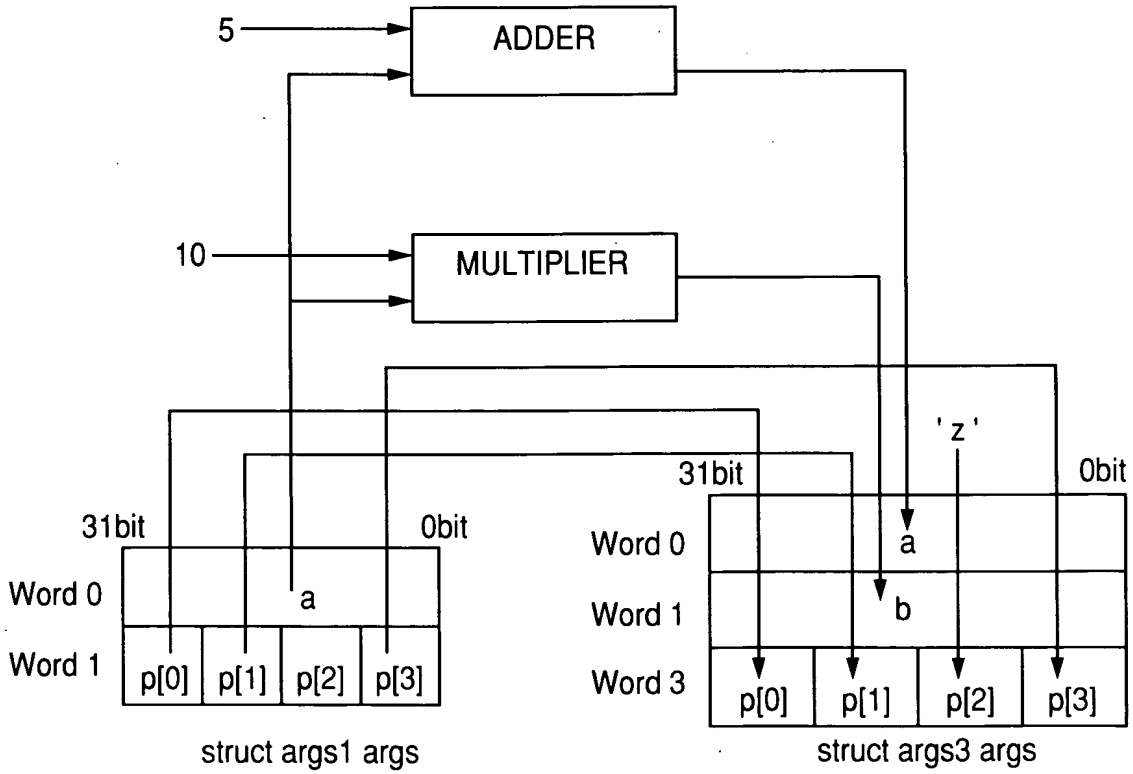


FIG. 11

FUNCTION NUMBER	ARGUMENT STRUCTURE

1	int a		char p[3]	

FIG. 12

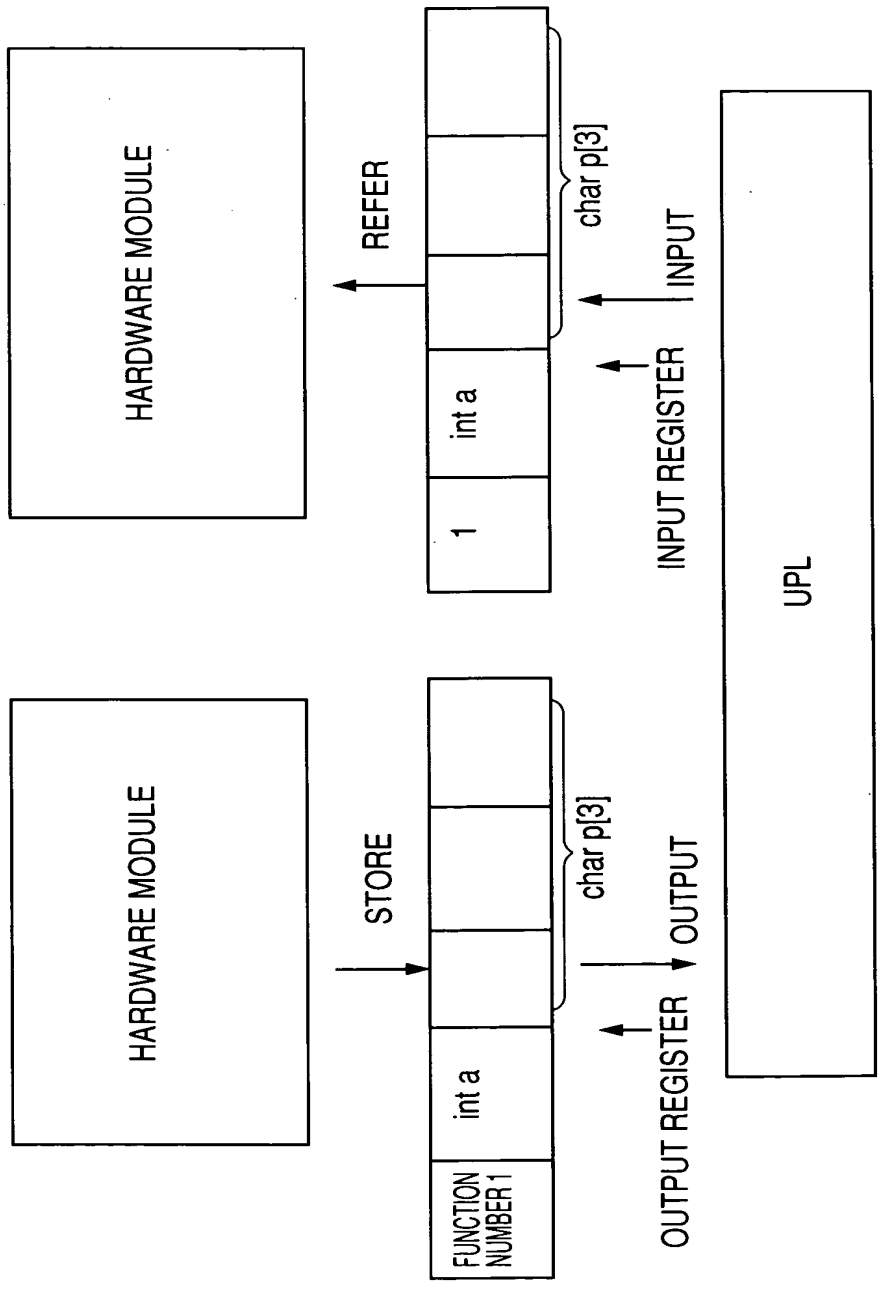

HARDWARE MODULE
REFER
1
int a
char p[3]
INPUT REGISTER
INPUT
UPL
HARDWARE MODULE
STORE
FUNCTION NUMBER 1
int a
char p[3]
OUTPUT REGISTER
OUTPUT

FIG. 13

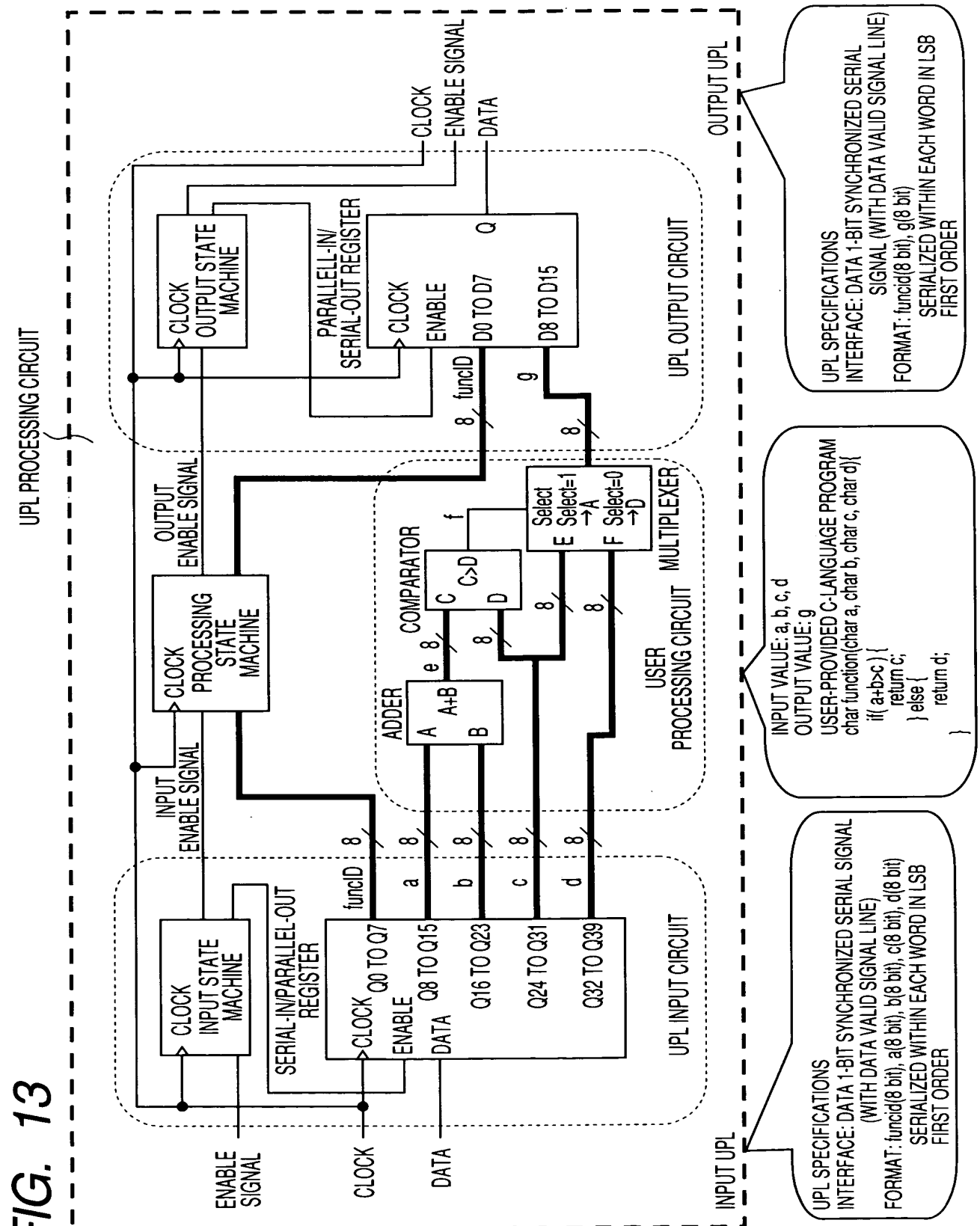

UPL PROCESSING CIRCUIT
INPUT UPL
OUTPUT UPL
ENABLE SIGNAL
CLOCK
DATA
CLOCK
ENABLE SIGNAL
DATA
INPUT ENABLE SIGNAL
OUTPUT ENABLE SIGNAL
CLOCK INPUT STATE MACHINE
SERIAL-IN/PARALLEL-OUT REGISTER
CLOCK
ENABLE
DATA
funcID
Q0 TO Q7
Q8 TO Q15
Q16 TO Q23
Q24 TO Q31
Q32 TO Q39
UPL INPUT CIRCUIT
CLOCK PROCESSING STATE MACHINE
ADDER
A
B
A+B
e
COMPARATOR
C
D
C>D
f
Select
E Select=1 →A
F Select=0 →D
MULTIPLEXER
USER PROCESSING CIRCUIT
CLOCK OUTPUT STATE MACHINE
PARALLELL-IN/ SERIAL-OUT REGISTER
CLOCK
ENABLE
D0 TO D7
D8 TO D15
Q
funcID
g
UPL OUTPUT CIRCUIT
UPL SPECIFICATIONS
INTERFACE: DATA 1-BIT SYNCHRONIZED SERIAL SIGNAL (WITH DATA VALID SIGNAL LINE)
FORMAT: funcid(8 bit), a(8 bit), b(8 bit), c(8 bit), d(8 bit)
SERIALIZED WITHIN EACH WORD IN LSB FIRST ORDER
INPUT VALUE: a, b, c, d
OUTPUT VALUE: g
USER-PROVIDED C-LANGUAGE PROGRAM
char function(char a, char b, char c, char d){
if(a+b>c) {
return c;
} else {
return d;
}
UPL SPECIFICATIONS
INTERFACE: DATA 1-BIT SYNCHRONIZED SERIAL SIGNAL (WITH DATA VALID SIGNAL LINE)
FORMAT: funcid(8 bit), g(8 bit)
SERIALIZED WITHIN EACH WORD IN LSB FIRST ORDER

FIG. 14

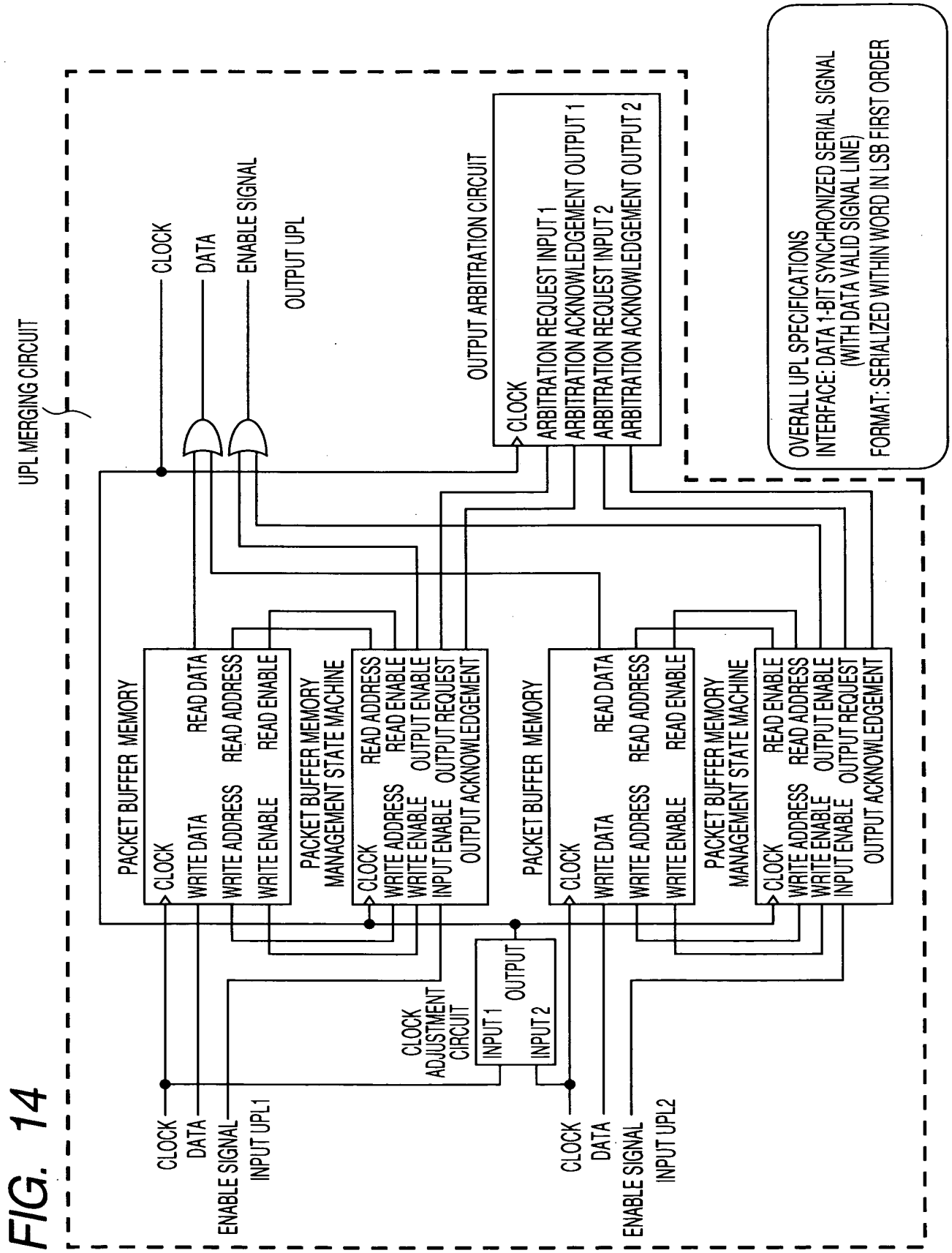

UPL MERGING CIRCUIT
CLOCK
DATA
ENABLE SIGNAL
OUTPUT UPL
OUTPUT ARBITRATION CIRCUIT
CLOCK
ARBITRATION REQUEST INPUT 1
ARBITRATION ACKNOWLEDGEMENT OUTPUT 1
ARBITRATION REQUEST INPUT 2
ARBITRATION ACKNOWLEDGEMENT OUTPUT 2
OVERALL UPL SPECIFICATIONS
INTERFACE: DATA 1-BIT SYNCHRONIZED SERIAL SIGNAL
(WITH DATA VALID SIGNAL LINE)
FORMAT: SERIALIZED WITHIN WORD IN LSB FIRST ORDER
PACKET BUFFER MEMORY
CLOCK
WRITE DATA
WRITE ADDRESS
WRITE ENABLE
READ DATA
READ ADDRESS
READ ENABLE
PACKET BUFFER MEMORY MANAGEMENT STATE MACHINE
CLOCK
WRITE ADDRESS
WRITE ENABLE
INPUT ENABLE
READ ADDRESS
READ ENABLE
OUTPUT ENABLE
OUTPUT REQUEST
OUTPUT ACKNOWLEDGEMENT
CLOCK ADJUSTMENT CIRCUIT
INPUT 1
INPUT 2
OUTPUT
CLOCK
DATA
ENABLE SIGNAL
INPUT UPL1
CLOCK
DATA
ENABLE SIGNAL
INPUT UPL2

FIG. 15

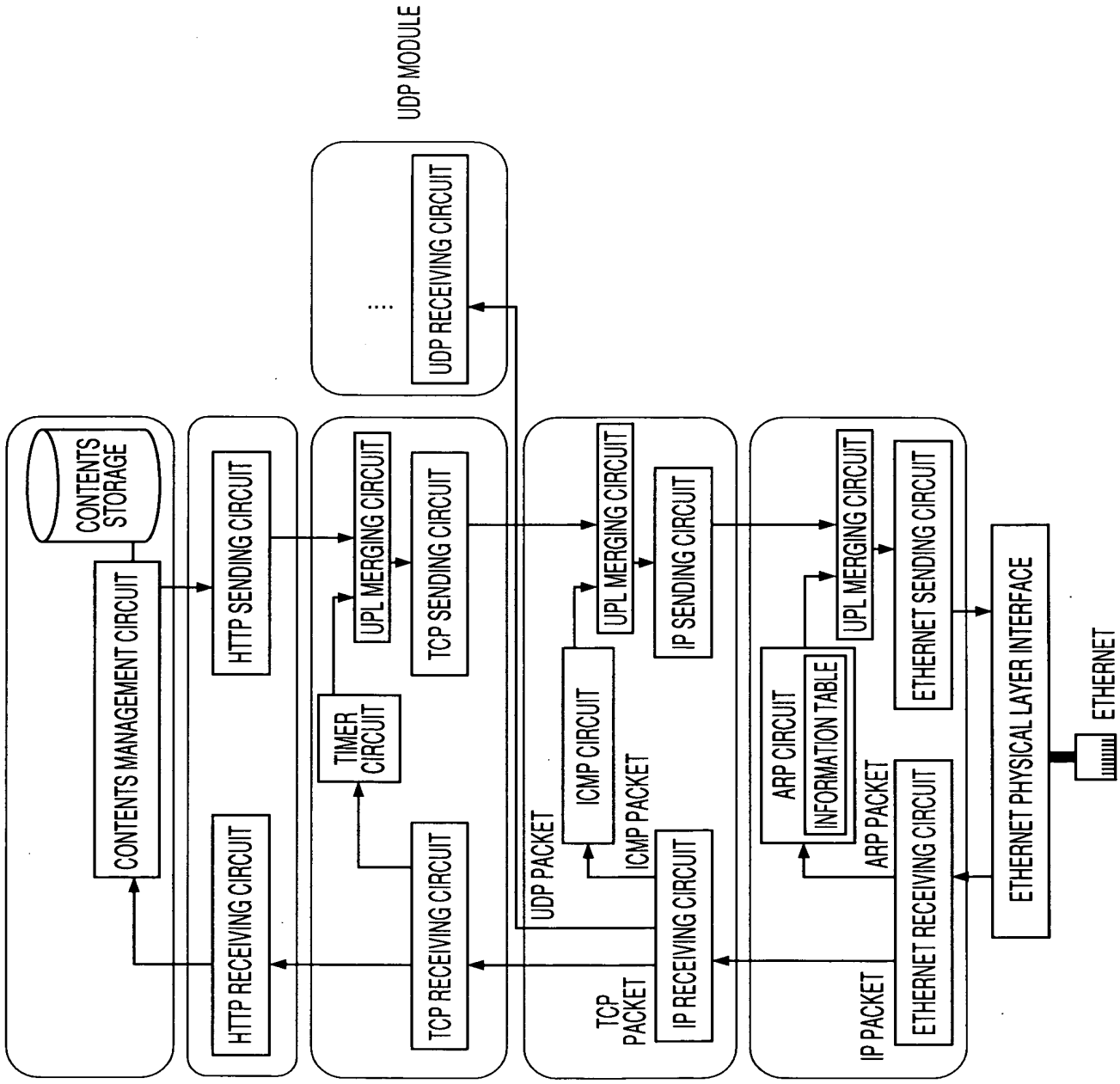
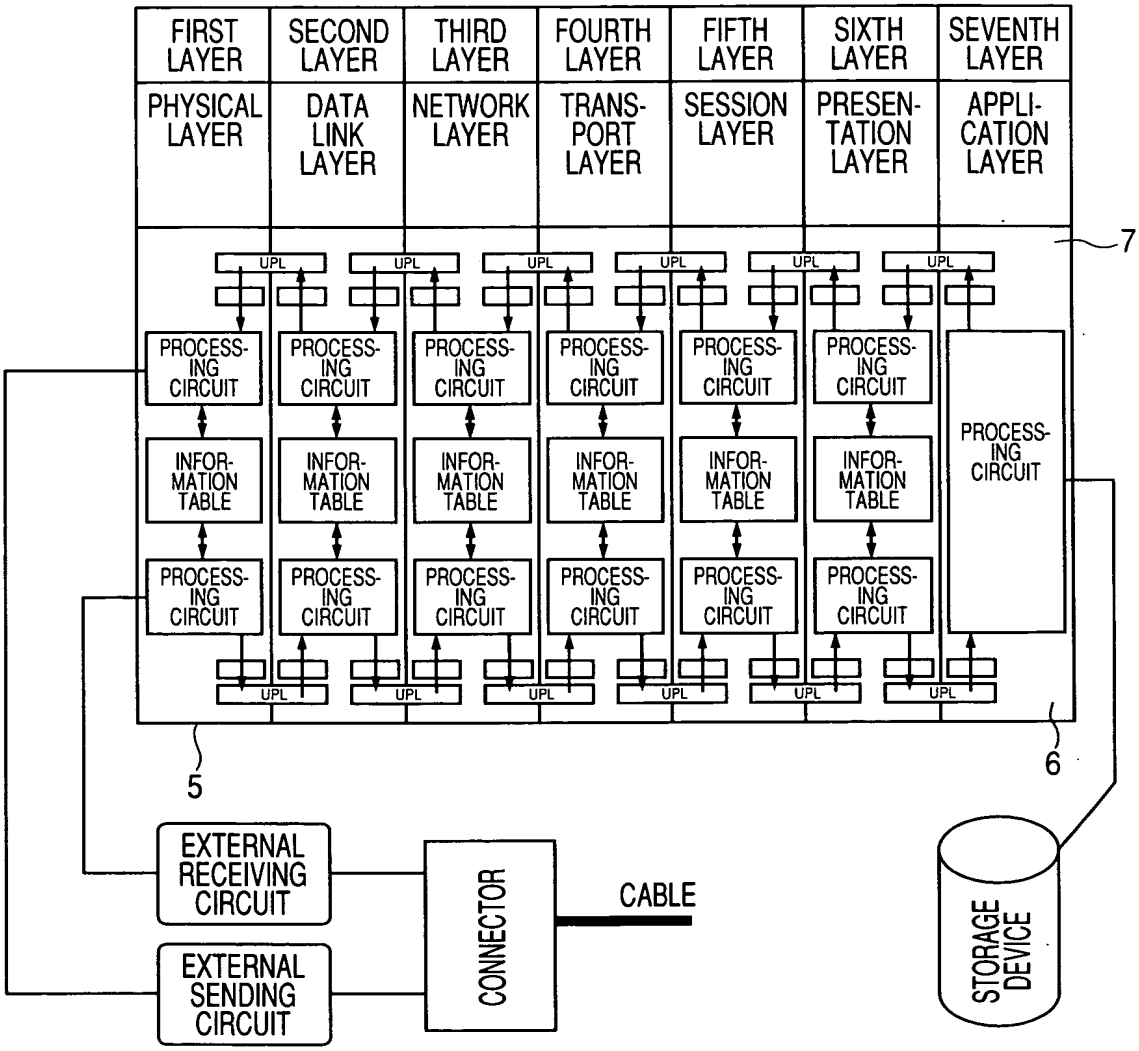

CONTENTS MODULE
HTTP MODULE
TCP MODULE
IP MODULE
ETHERNET MODULE
UDP MODULE
CONTENTS STORAGE
CONTENTS MANAGEMENT CIRCUIT
HTTP RECEIVING CIRCUIT
HTTP SENDING CIRCUIT
TIMER CIRCUIT
UPL MERGING CIRCUIT
TCP SENDING CIRCUIT
TCP RECEIVING CIRCUIT
UDP PACKET
ICMP CIRCUIT
ICMP PACKET
UPL MERGING CIRCUIT
IP SENDING CIRCUIT
TCP PACKET
IP RECEIVING CIRCUIT
ARP CIRCUIT
INFORMATION TABLE
ARP PACKET
UPL MERGING CIRCUIT
ETHERNET SENDING CIRCUIT
IP PACKET
ETHERNET RECEIVING CIRCUIT
ETHERNET PHYSICAL LAYER INTERFACE
ETHERNET
UDP RECEIVING CIRCUIT

FIG. 16

(a)


FIRST LAYER
PHYSICAL LAYER
SECOND LAYER
DATA LINK LAYER
THIRD LAYER
NETWORK LAYER
FOURTH LAYER
TRANS-PORT LAYER
FIFTH LAYER
SESSION LAYER
SIXTH LAYER
PRESEN-TATION LAYER
SEVENTH LAYER
APPLI-CATION LAYER
UPL
PROCESS-ING CIRCUIT
INFOR-MATION TABLE
7
5
6
EXTERNAL RECEIVING CIRCUIT
EXTERNAL SENDING CIRCUIT
CONNECTOR
CABLE
STORAGE DEVICE

(b)

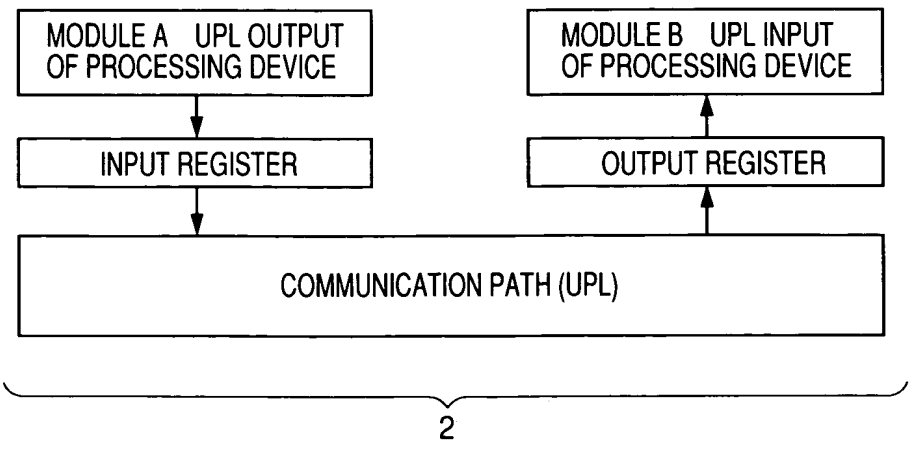

MODULE A UPL OUTPUT OF PROCESSING DEVICE
MODULE B UPL INPUT OF PROCESSING DEVICE
INPUT REGISTER
OUTPUT REGISTER
COMMUNICATION PATH (UPL)
2

FIG. 17

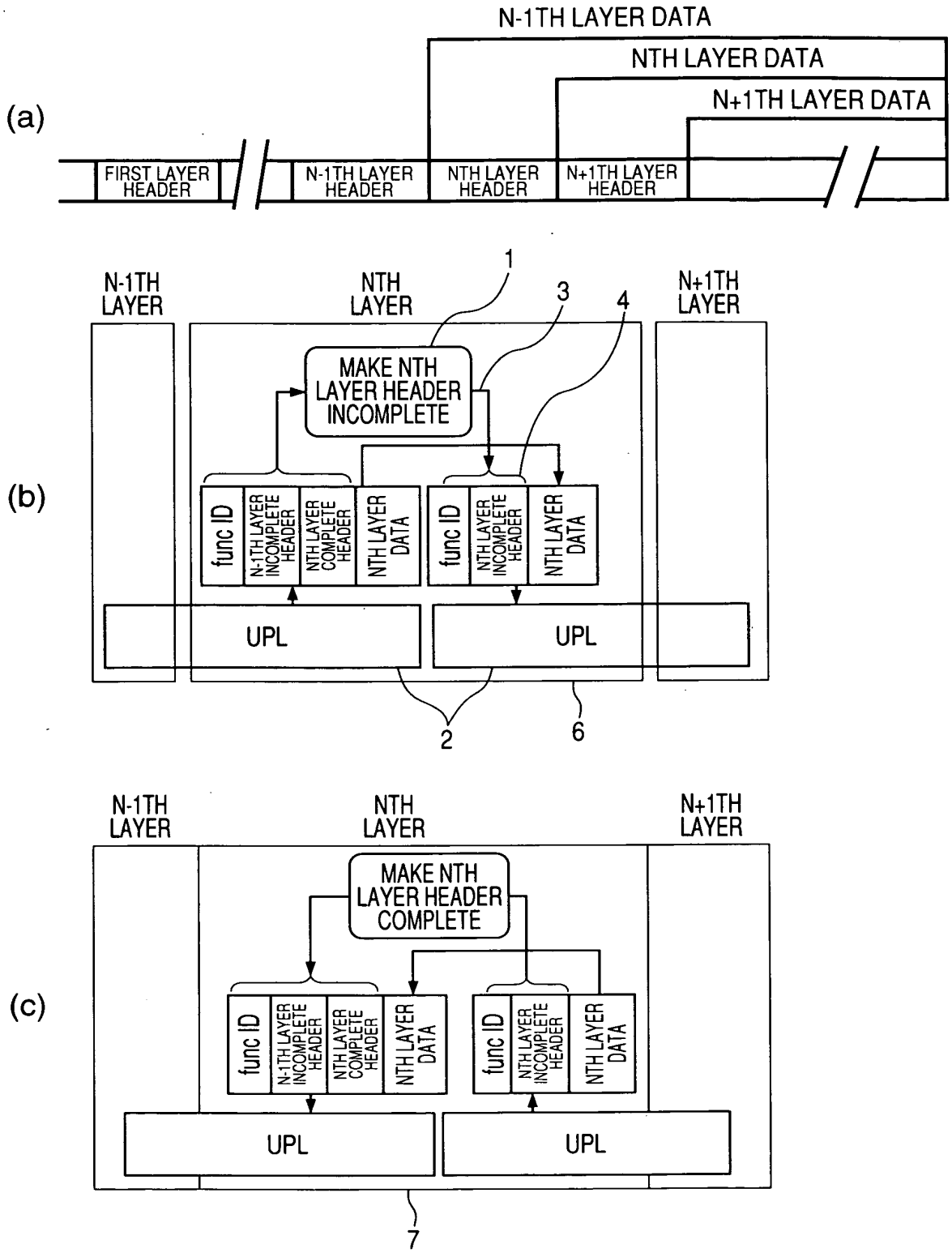

(a)
N-1TH LAYER DATA
NTH LAYER DATA
N+1TH LAYER DATA
FIRST LAYER HEADER
N-1TH LAYER HEADER
NTH LAYER HEADER
N+1TH LAYER HEADER
(b)
N-1TH LAYER
NTH LAYER
N+1TH LAYER
1
3
4
MAKE NTH LAYER HEADER INCOMPLETE
func ID
N-1TH LAYER INCOMPLETE HEADER
NTH LAYER COMPLETE HEADER
NTH LAYER DATA
UPL
2
6
(c)
N-1TH LAYER
NTH LAYER
N+1TH LAYER
MAKE NTH LAYER HEADER COMPLETE
7

FIG. 18

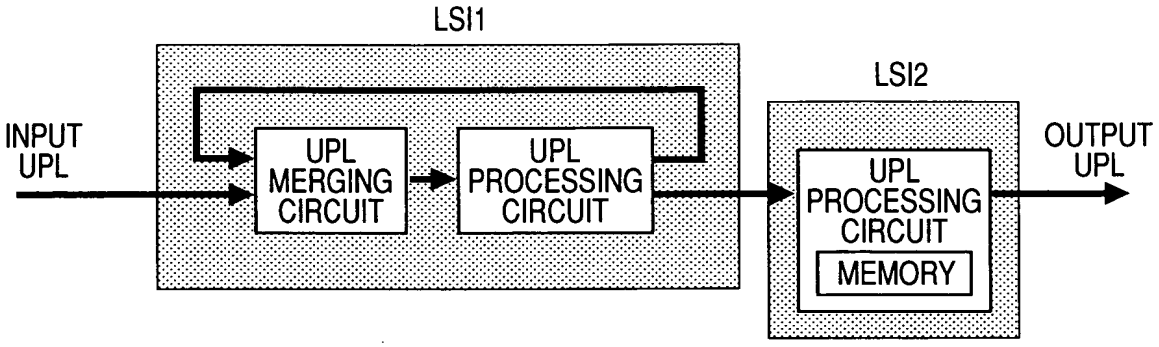

LSI1
LSI2
INPUT UPL
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT
UPL PROCESSING CIRCUIT
MEMORY
OUTPUT UPL

FIG. 19

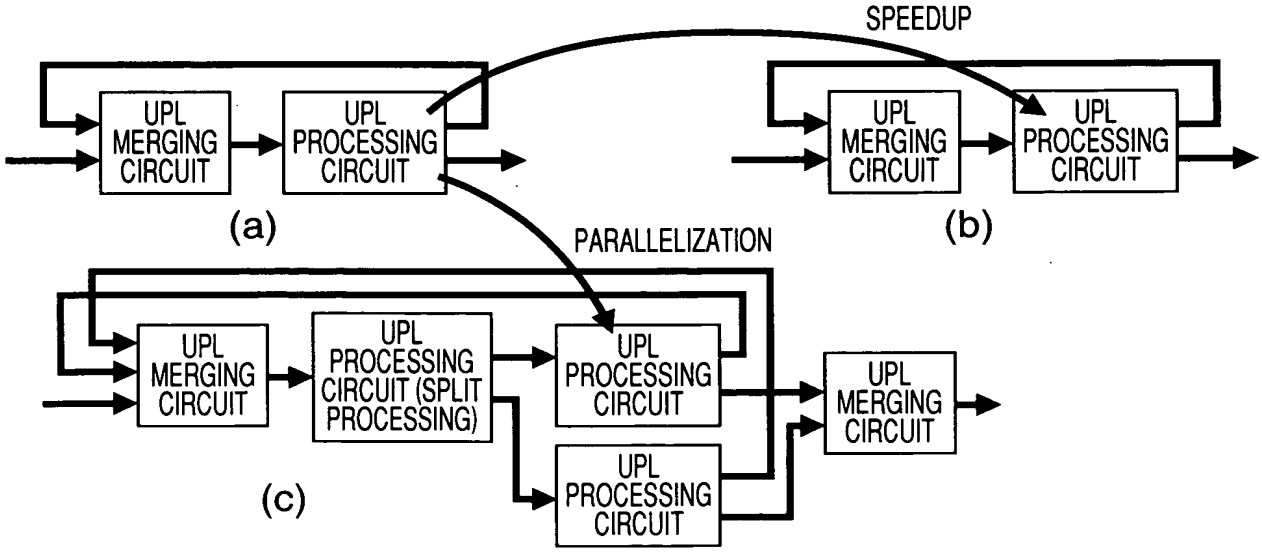

SPEEDUP
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT
(a)
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT
(b)
PARALLELIZATION
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT (SPLIT PROCESSING)
UPL PROCESSING CIRCUIT
UPL PROCESSING CIRCUIT
UPL MERGING CIRCUIT
(c)

FIG. 20

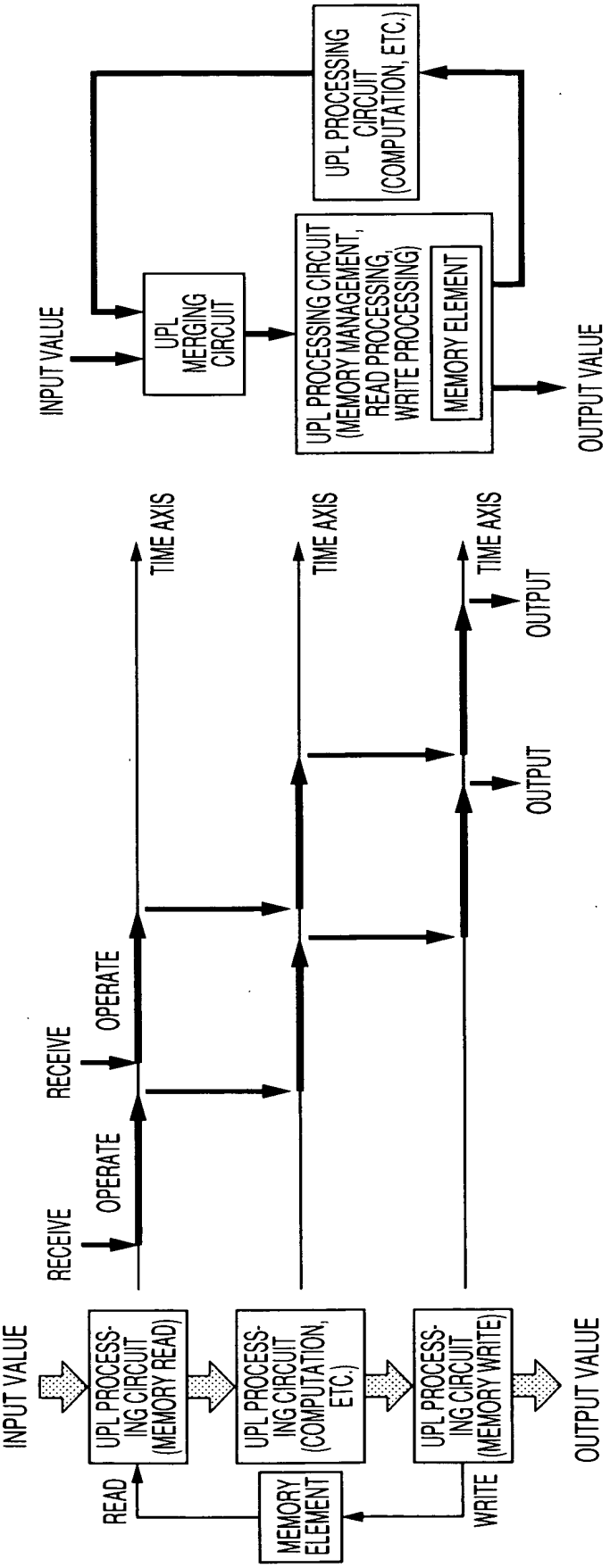

INPUT VALUE
UPL PROCESS-ING CIRCUIT (MEMORY READ)
UPL PROCESS-ING CIRCUIT (COMPUTATION, ETC.)
UPL PROCESS-ING CIRCUIT (MEMORY WRITE)
OUTPUT VALUE
READ
WRITE
MEMORY ELEMENT
RECEIVE
OPERATE
RECEIVE
OPERATE
TIME AXIS
TIME AXIS
TIME AXIS
OUTPUT
OUTPUT
INPUT VALUE
UPL MERGING CIRCUIT
UPL PROCESSING CIRCUIT (MEMORY MANAGEMENT, READ PROCESSING, WRITE PROCESSING)
MEMORY ELEMENT
UPL PROCESSING CIRCUIT (COMPUTATION, ETC.)
OUTPUT VALUE

FIG. 21

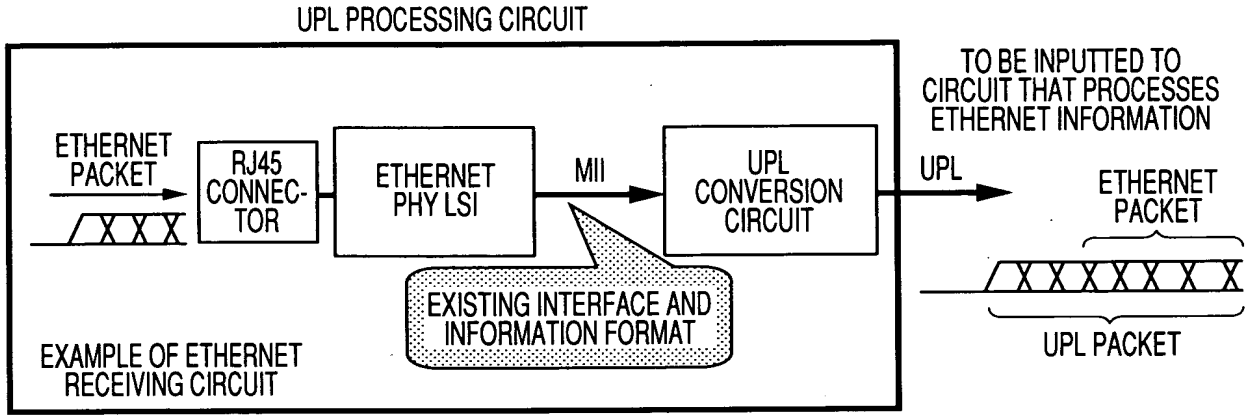

UPL PROCESSING CIRCUIT
ETHERNET PACKET
RJ45 CONNEC-TOR
ETHERNET PHY LSI
MII
UPL CONVERSION CIRCUIT
EXISTING INTERFACE AND INFORMATION FORMAT
EXAMPLE OF ETHERNET RECEIVING CIRCUIT
TO BE INPUTTED TO CIRCUIT THAT PROCESSES ETHERNET INFORMATION
UPL
ETHERNET PACKET
UPL PACKET

FIG. 22

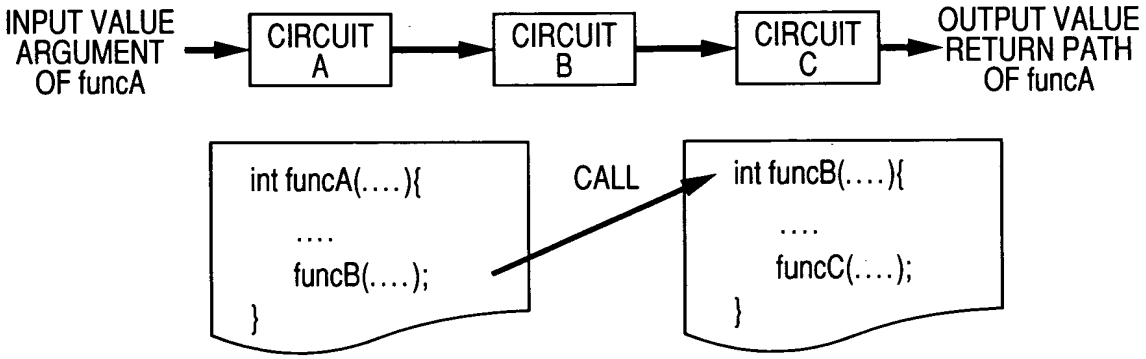

INPUT VALUE
ARGUMENT
OF funcA
CIRCUIT
A
CIRCUIT
B
CIRCUIT
C
OUTPUT VALUE
RETURN PATH
OF funcA
int funcA(....){
....
funcB(....);
}
CALL
int funcB(....){
....
funcC(....);
}

FIG. 23

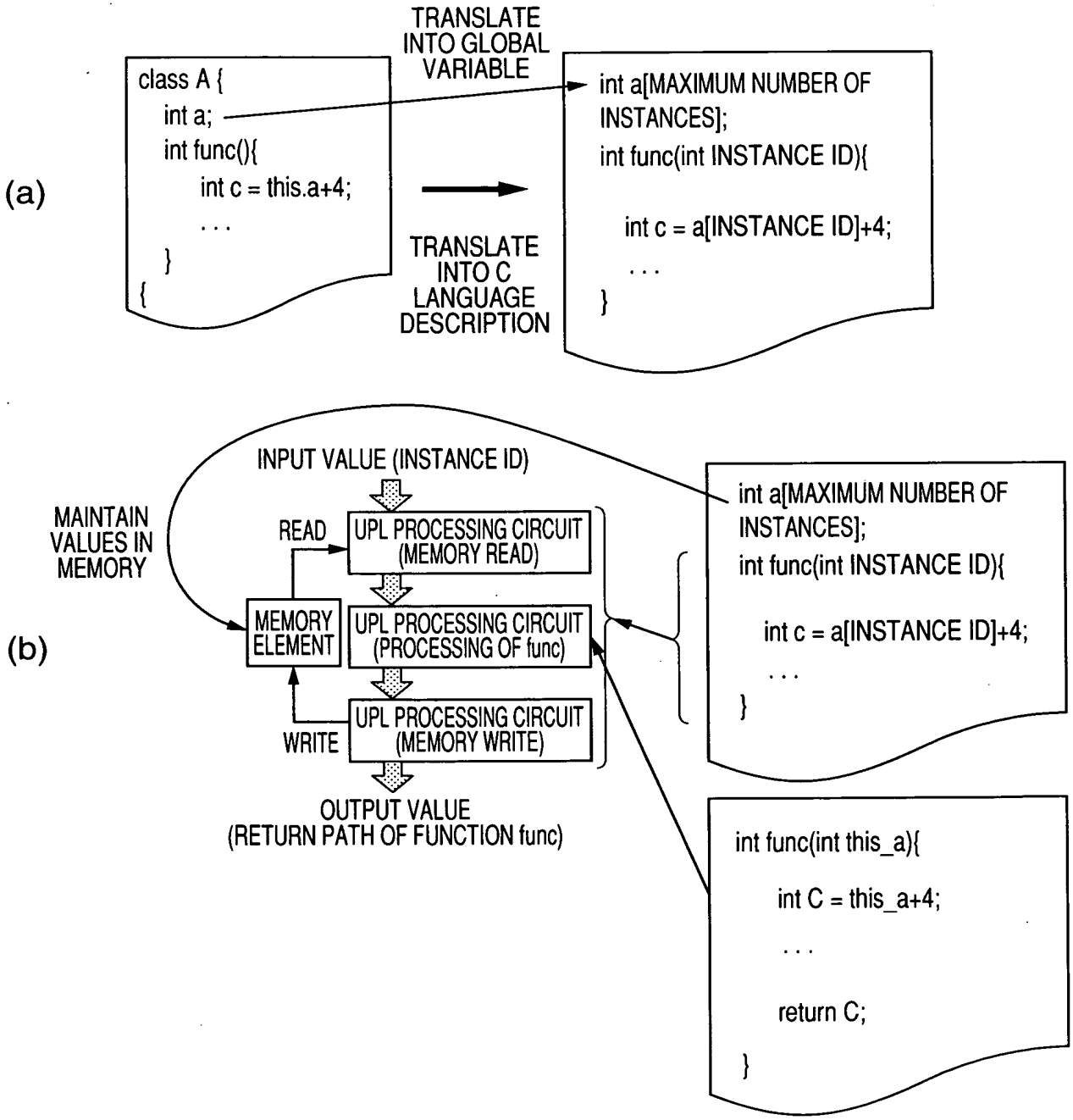
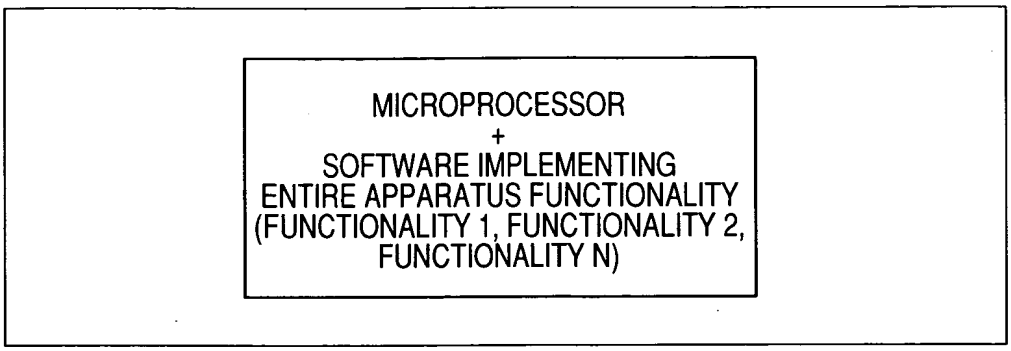
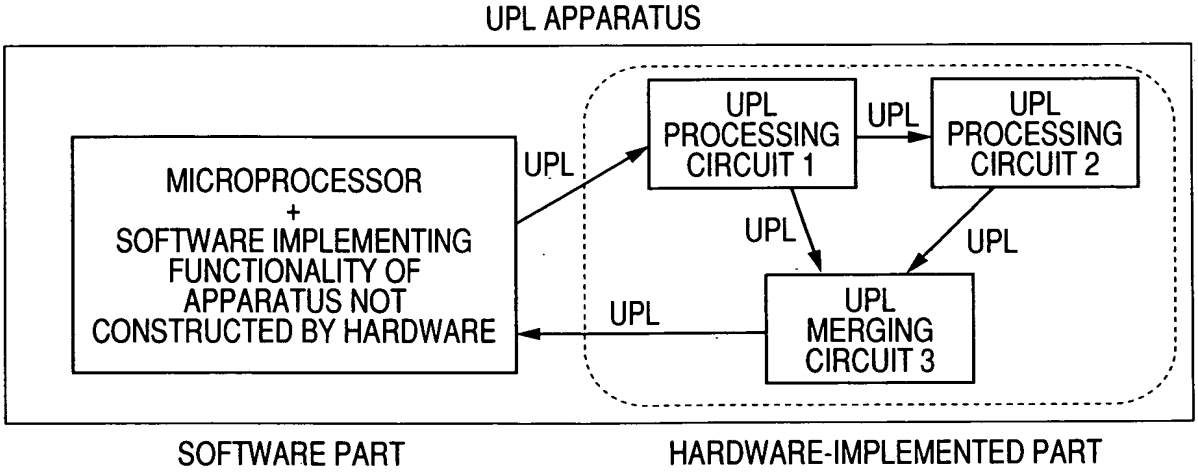
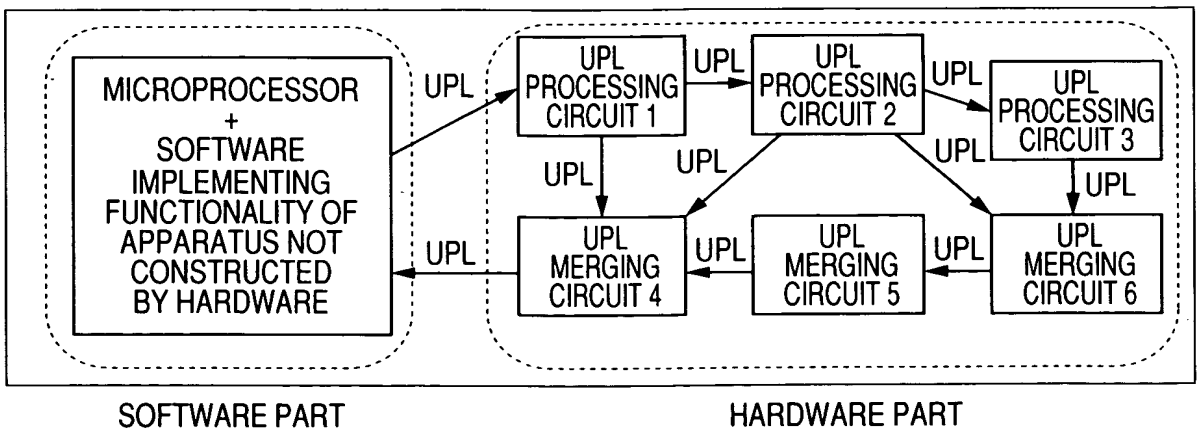

(a)
class A {
int a;
int func(){
int c = this.a+4;
. . .
}
{
TRANSLATE INTO GLOBAL VARIABLE
TRANSLATE INTO C LANGUAGE DESCRIPTION
int a[MAXIMUM NUMBER OF INSTANCES];
int func(int INSTANCE ID){
int c = a[INSTANCE ID]+4;
. . .
}
(b)
INPUT VALUE (INSTANCE ID)
MAINTAIN VALUES IN MEMORY
READ
UPL PROCESSING CIRCUIT (MEMORY READ)
MEMORY ELEMENT
UPL PROCESSING CIRCUIT (PROCESSING OF func)
WRITE
UPL PROCESSING CIRCUIT (MEMORY WRITE)
OUTPUT VALUE (RETURN PATH OF FUNCTION func)
int a[MAXIMUM NUMBER OF INSTANCES];
int func(int INSTANCE ID){
int c = a[INSTANCE ID]+4;
. . .
}
int func(int this_a){
int C = this_a+4;
. . .
return C;
}

FIG. 24

(a) · INITIAL DEVELOPMENT STAGE



(b) · HARDWARE IMPLEMENTATION STAGE



(c) · DEVELOPMENT COMPLETION STAGE